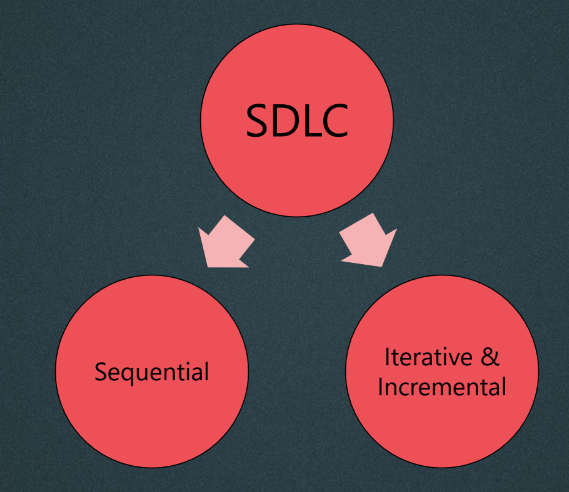
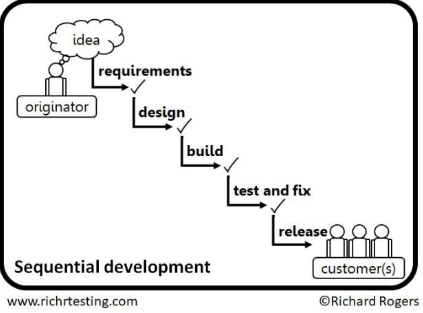
Kurs Linki: [The Complete 2023 Software Testing Bootcamp | Udemy](https://www.udemy.com/course/testerbootcamp/)

**System Development Life Cycle (Yazılım Geliştirme Yaşam Döngüsü)**

🡺 SDLS iki kısma ayrılır. Birincisi sıralı (**Sequential**) geliştirme, ikincisi yinelemeli (**Iterative**) ve artımlı (**Incremental**) geliştirmedir.



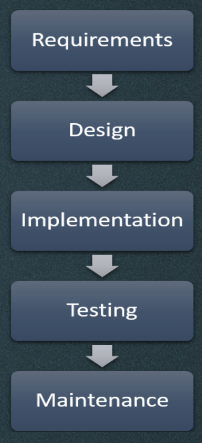
**1)** Sıralı geliştirme:



🡺 Sıralı geliştirmede adımlar birbirini takip eder. Teorik olarak bir adımın başlaması için önceki adımın bitmesi gerekir. Bu duruma göre adımlar arasında çakışma olmaz. Uygulama kısmında ise projenin bir kısmı tasarlarken, önceden tasarlanan kısmın testi yapılabilir. Aynı şekilde projeyi geliştirme devam ederken müşteri farklı bir özellik istediğinde önceki adımlara geri dönülebilir. Bu şekilde istisnalar bulunabilir.

🡺 Sıralı geliştirme iki kısma ayrılır. Şelale (**Waterfall**) ve **V-Model**.

**1.1)** Şelale (Waterfall) geliştirme:



🡺 Bu modelde test işlemleri, bütün işlemler bittikten sonra başlar.

🡺 Şelale modeli denmesinin sebebi, şelalelerde akan suyun bir daha geri dönmemesinden gelmektedir. Uygulama gereksinimleri toplanır, tasarlanır, uygulanır, test edilir ve yayınlanır. Eğer müşteri farklı bir özellik daha eklemek isterse, o zaman projeye bakım (**Maintenance**) yapılır.

**1.2)** V-Model geliştirme:

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

🡺 V-Model geliştirmede gereksinimler Kullanıcı gereksinimleri ve yazılım gereksinimleri (veya teknik gereksinimler) olarak ikiye ayrılır. Tasarım kısmı, yüksek seviye tasarım ve düşük seviye tasarım olarak iki bölüme ayrılır. Projeyi geliştirmeye sol taraftan başlanır. V-Model geliştirmede ilk 5 adım şelale geliştirmenin ilk 3 adımı ile aynıdır.

🡺 V- Model geliştirmede test kısmı 4 bölüme ayrılır. Bunlar birim testi, entegrasyon testi, sistem testi ve kabul testidir.

**2)** Yinelemeli (Iterative) ve artımlı (Incremental) geliştirme:

metin, farklı içeren bir resim

Açıklama otomatik olarak oluşturuldu

🡺 Yinelemeli (**Iterative**) geliştirmede proje iterasyonlara (Birbirini takip eden işlemler) bölünür. Mesela yukarıdaki örnekte, ressam ilk olarak resmi sadece kara kalemle çizmiş daha sonra adımlar ilerleyince renklendirmiş.

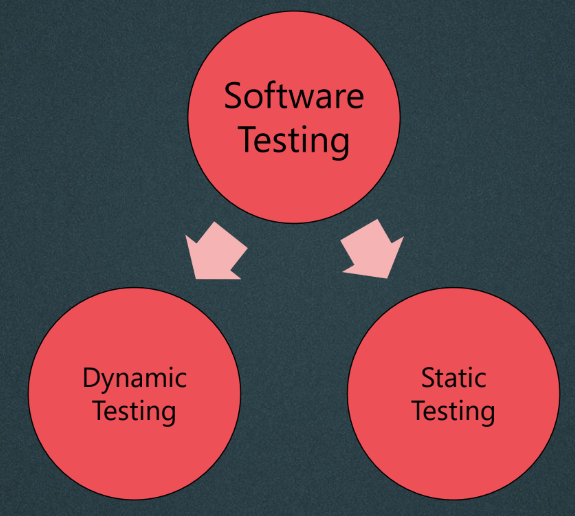
🡺 Artımlı (**Incremental**) geliştirmede proje bölümlere ayrılır. Bir bölümün gereksinimleri toplanır, tasarlanır, uygulanır ve test edilir. Daha sonra diğer bölüme geçilir.

🡺 Projemizde aynı anda hem artımlı geliştirmeyi hem de yinelemeli geliştirmeyi kullanabiliriz. Yani artımlı geliştirmede ayırdığımız bölümleri iterasyonlara ayırıp geliştirmemizi yapabiliriz. Bu duruma en iyi örnek **Scrum’lardır**.

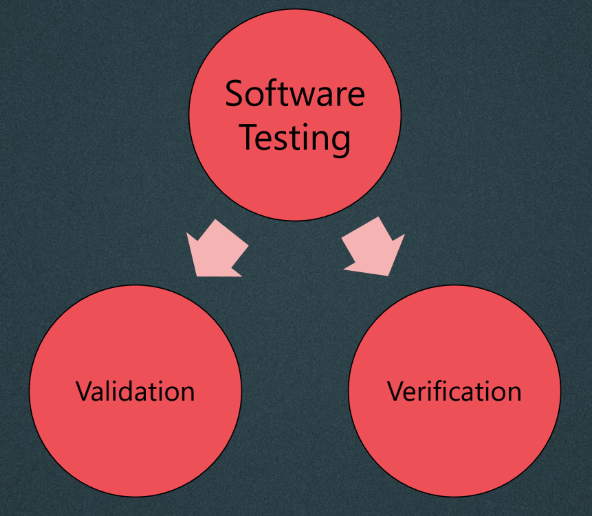
***NOT:*** Scrum nedir 🡪 [SCRUM NEDİR?. Zaman içerisinde projelerin daha büyük ... - Medium](https://medium.com/@secilcor/scrum-nedi%CC%87r-6a4326951dd8)  
***NOT:*** Scrum nedir video anlatım 🡪 [Introduction to Scrum - 7 Minutes - YouTube](https://www.youtube.com/watch?v=9TycLR0TqFA)

**Software Testing (Yazılım Testi)**

🡺 Yazılım testi, Dinamik (**Dynamic**) test ve Statik (**Static**) test olarak iki kısma ayrılabilir. Dinamik testte yazılımı çalıştırıp test ederiz. Statik testlerde yazılımı çalıştırmayız. Gereksinimleri gözden geçirebiliriz, tasarımı gözden geçirebiliriz, kodumuzu gözden geçirebiliriz kısacası her türlü gözden geçirmeyi yapabiliriz.



🡺 Ayrıca yazılım testi, Doğrulama (**Validation**) ve Onaylama (**Verification**) olarak 2 kısma ayrılabilir. Doğrulama (**Validation**) doğru ürünü oluşturduğumuz anlamına gelir. Kullanıcı bu ürünü kullandığında memnun olacak. Onaylama (**Verification**) ürünü doğru bir şekilde oluşturduğumuz anlamına gelir.



🡺 Test süreci, yazılımı test etmek için uyguladığımız adımlardır. Test süreci 3 adıma ayrılır. Plan(Planlama), Design(Tasarım) ve Execution(Yürütme). Yazılım testi için tek bir evrensel test süreci yoktur.

diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

🡺 2018 ISTQB müfredatına göre test süreci 7 faaliyetten oluşur. İlk olarak test planımızı yazarız. Daha sonra projede ilerlerken, ilerlememizi izlemeye başlarız. Gerçek ilerlememizi, planlama aşamasında yazdığımız beklenen ilerleme ile karşılaştırırız. Eğer bir fark var ise Control(Kontrol) faaliyetlerini gerçekleştiririz. Daha sonra gereksinimlerimizi okuyarak, analiz ederiz. Bu aşamadan sonra test senaryoları yazmaya başlarız. Bir sonraki adımda ise testlerimizi uygulamaya başlarız. Daha sonra farklı platformlarda testleri yürütürüz, hataları buluruz, sonuçları raporlarız, yeniden test ederiz ve regresyon testleri gerçekleştiririz. En son adımda ise test işlemini tamamlarız, test ilerleme raporları ve test özet raporları yazarız.

dikdörtgen içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Test Levels (Test Seviyeleri)**

🡺 4 Ana test seviyesi vardır.

**1) Unit Test – Component Test (Birim testi – Modül (Bileşen) testi):** Ayrı olarak test edilenleri test eder. Kod modülleri, Kod sınıfları vb. Çoğu zaman bu test türünden sorumlu olan kişi geliştiricidir.

**2) Integration Test (Entegrasyon testi):** Entegrasyon testi, bileşenler veya sistemler arasındaki etkileşimlere odaklanır. İki tür entegrasyon testi vardır. Bunlar Component Integration (Bileşen entegrasyon) ve System Integration (Sistem entegrasyon) testleridir. Bileşen entegrasyon testi, bileşenler arasındaki entegrasyona odaklanır. Bu test türü de geliştirici tarafından gerçekleştirilir. Sistem entegrasyon testi ise backend, database, frontend vb sistemlerin birbirine entegre edilmesine denir. Sistem entegrasyon testini, test uzmanı gerçekleştirir.

**3) System Test (Sistem testi):** Sistem testi, sistemi bir bütün olarak test etmeye denir. Bu test uzmanları için en önemli test seviyesidir. Bu seviyede uygulamamızı gerçek ortama benzer bir ortamda test etmeye çalışırız. Sistem testi sırasında mümkün olduğunca çok hata bulmaya çalışırız. Bu test seviyesi, test uzmanının sorumluluğundadır.

**4) Acceptance Test (Kabul testi):** Sistem testini gerçekleştirip hataları bulduktan sonra, kabul testini uyguluyoruz. Bu testin amacı, uygulamamızın doğru performans gösterip göstermediğinden emin olmaktır. Hata bulmak bu test seviyesinin ana amacı değildir. Bu test seviyesi, çoğu zaman kullanıcılar ve paydaşlar tarafından yapılır. Alpha (Alfa) testi ve Beta testi olarak adlandırılan bir kabul testimiz vardır. Alfa testi, müşterileri şirkete getirdiğimiz ve tesislerimizde test etmelerini sağladığımız anlamına gelir. Beta testi ise müşterilerimizin uygulamamızı kendi evlerinde test etmelerini sağladığımız anlamına gelir.

**Testing Types (Test Türleri)**

🡺 Onlarca test türü vardır. Bazıları şu şekildedir;

**1-) Functional Testing (Fonksiyonel Testler):** Yazılımın ana işlevlerini test eder. Örnek olarak Facebook’un oturum açma işlevi. Bu test türünde cevap olarak genellikle evet veya hayır cevabını veririz.

**2-) Non-functional Testing (Fonksiyonel Olmayan Testler):** Bu testler sistemin nasıl çalıştığını test eder. Örnek olarak Facebook hesabına girerken yeterince hızlı giriyor mu?. Bu işlevin yanıt süresi iyi mı? Performans testi bu test türüne girer. Bu test türünde evet veya hayır cevabını veremeyiz. Örneğin bu yazılım hızlı ya da bu yazılım yavaş diyebiliriz. Çoğu zaman bir aralıkla cevap veriyoruz, yazılımı aynı anda kullanan 1000 eşzamanlı kullanıcıyla yük testi gibi bir şey gerçekleştirdiğimizi söylüyoruz. Ve bu durumda yanıt süresi, kullanıcının ortalama yanıt süresi örneğin 3 saniye ve kullanıcıların %99’u bu test durumunu geçti gibi. Kullanılabilirlik testi de bu türe girer. Kullanıcı arayüzü yeterince çekici mi? Bu duruma evet ya da hayır diyemeyiz. Bu bir bakış açısıdır.

**3-) Black-Box Testing (Kara Kutu Testleri):** Kara kutu testi yazılımın iç yapısına bakmadan test ettiğimiz anlamına gelir. Örneğin bir mobil uygulamam varsa, girdi sağlayacağım ve çıktıyı bekleyeceğim.

**4-) White-Box Testing (Beyaz Kutu Testleri):** Beyaz kutu testlerinde yazılıma girdi sağlarız ve aynı zamanda yazılımın iç yapısına, kodda, veritabanında, API’lerde neler olduğuna bakarız.

**5-) Dynamic Testing (Dinamik Test):** Yazılımı çalıştırdığımız anlamına gelir.

**6-) Static Testing (Statik Test):** Yazılımda hiçbir şey çalıştırmadığımız halde test etmektir. Örneğin gereksinimlerin gözden geçirilmesi, kullanıcı hikayelerinin gözden geçirilmesi, geliştiricinin yazdığı kodu incelemek, kullanıcı arayüzünün bir prototipini gözden geçirmek ve bazı notlar vermek gibi.

**7-) Retesting – Confirmation Testing (Yeniden Test Etme – Onay (Doğrulama) Testi):** Geliştiricinin kodunda bir hata bulduk ve bu hatayı geliştiriciye rapor ettik. Ardından geliştirici bu hatayı düzeltti ve bize tekrar gönderdi. Biz bu durumu tekrar uygularız ve sorunun çözülüp çözülmediğine bakarız.

**8-) Regression Testing (Regresyon Testi):** Geliştirici yeni bir özellik ekler veya özellik kaldırabilir. Bu durumun, diğer işlevleri etkileyip etkilemediğini test ederiz. Retesting’de sadece hata olan kısmı test ediyorduk. Bu test türünde ise yazılımın değişmeyen kısımlarını test edeceğiz.

**9-) Smoke Testing (Duman Testi):** Bu test türü, geliştirilen yazılımın fonksiyonlarının doğru çalışıp çalışmadığını kontrol etmek amacıyla yapılan yazılım test sürecidir. Yazılım fonksiyonlarını test etmek için her yapı üzerinde yürütülen bir dizi testten oluşur. Amaç, hataları erkenden tespit edip önleyerek maliyet ve zaman kaybının önüne geçilmesidir. Duman testleri, sistem kararlılığını ve gereksinimlere uygunluğunu göstermek için tasarlanmıştır. Sistemde login işlemi varsa bunlar kontrol edilebilir. Login olunur (login olunabiliyor mu diye), şifre sıfırlama denenir süreç işliyor mu diye ya da sayfalar arası gezilir (sayfalar açılıyor mu diye), profil güncellenir (güncellenebiliyor mu diye) vs. En temel özellikler çalışıyorsa yapılan geliştirmenin detaylı testinin yapılması için sistem test ekibine teslim edilir.

**Test Scenario and Test Case (Test Senaryosu ve Test Durumu)**

🡺 Test scenario (Test senaryosu) neyin test edileceğidir. Buna Test Condition (Test Koşulu) veya Test Possibility (Test Olasılığı) de denmektedir. Test case (Test Durumu) ise test edilecek senaryonun nasıl test edileceğidir. Örneğin Gmail uygulamasının oturum açma sayfasını test ettiğimizi varsayalım. Buradaki test senaryosu, oturum açma işlevini doğrulamaktır. Test case ise bunu hangi adımlar ile yapacağımızdır.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Test Scenario Writing (Test Senaryosu Yazma)**

**1-)** Test edilen sisteme (SUT) (System Under Test) ilişkin, Business Requirement Specification (BRS) (İş Gereksinim Şartnamesi), Software Requirement Specification (SRS) (Yazılım Gereksinim Şartnamesi) ve Functional Requirement Specification (FRS) (Fonksiyonel Gereksinim Şartnamesi) gereksinim belgelerini dikkatlice inceleyin.

**2-)** Her gereksinimi ayırın ve bu gereksinimler için hangi olası kullanıcı eylemlerinin test edilmesi gerektiğini belirtin.

**3-)** Test senaryolarını numaralandırın. Bu senaryoların, web sitesinin veya uygulamanın işleyişine dahil olan her kullanıcı akışını ve iş akışını kapsadığından emin olun.

**4-)** Test senaryolarını listeledikten sonra, her gereksinimin bir test senaryosuna eşlendiğinden emin olmak için bir **Traceability Matrix** (İzlenebilirlik Matrisi) oluşturun.

**5-)** Senaryoların bir gözetmen tarafından gözden geçirin ve ardından projeye dahil olan diğer paydaşlar tarafından gözden geçirilmesini sağlayın.

**Test Case (Test Durumu)**

🡺 Test case (Test durumu) uygulamamızı test ederken bize rehberlik eder. Test case ile ilgili olan, aşağıdaki terimleri bilmeliyiz.

**1-) Test Case Title (Test Durumu Başlığı):** Başlığın basit olması gerekmektedir. Başlığı okuduğumuzda neyi test etmemiz gerektiğini anlamalıyız.

**2-) Precondition (Ön Koşul):** Test case’i çalıştırmadan önce karşılanması gereken bir koşuldur. Örneğin giriş fonksiyonunu test etmemiz için önceden açılmış bir hesabımızın olması gerekir.

**3-) Test Steps (Test Adımları):** Test case’i yürütmek için gereken adımlardır. Bu adımlar açık ve bilgilendirici olmalıdır. Aşağıdaki resimde de gördüğünüz gibi, testi yapan kişiden belirli bir kullanıcı adı ve şifre istemedik. Ondan sadece geçerli bir kullanıcı adı ve şifre girmesini istedik. Buna **“High Level Test Case (Yüksek Seviye Test Durumu)“** denir. **“Low Level Test Case (Düşük Seviye Test Durumu)”** de ise belirli bir kullanıcı adı veya şifrenin girilmesi istenir. Örneğin kullanıcı adı Ahmet olsun, şifre 123 olsun gibi. Low level test case de tester bunları kullanarak test case i uygular. High level test case, yazılım teslimata hazır hale gelmeden yazılır. Low level test case ise yazılımımız teslimata hazır halde geldikten sonra yazılır. Örneğin eğer kullanıcı 99.99 TL’lik bir ürün alırsa indirim kazanacak mı? Bunun gibi durumları test ederiz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**4-) Expected Result (Beklenen Sonuç):** Test case sonucunda olması gereken işlemdir. Örneğin Giriş yapma test case i uygulanırken beklenen sonuç kullanıcının başarılı bir şekilde giriş yapabilmesidir.

**5-) Test Scenario - Test Suite (Test Senaryosu):** Test case’leri içerisinde bulunduran bir kavramdır. Örneğin test senaryomuz oturum açma iken test case’lerimiz facebook ile oturum açma veya gmail ile oturum açma olabilir. Test senaryoları, test case’leri kapsar.

**6-) Test Environment (Test Ortamı):** Test case’i test edeceğimiz ortamdır.

**7-) Actual Result (Gerçek Sonuç):** Test case sonucunda olan işlemdir. Diğer bütün işlemler test case yazarken (planlarken) önceden belirlenebilir. Fakat beklenen sonuç test case yürütülmeden yazılamaz.

**8-) Status (Durum):** Test case’in durumunu belirtir. 4 ana durum vardır. Bunlar;

**New (Yeni):** Test edilmeye hazır. Daha test edilmedi.

**Pass (Geçti):** Test case uygulandı ve beklenen sonuç ile gerçek sonuç aynı çıktı. Yani test başarılı.

**Fail (Kaldı):** Test case uygulandı ve beklenen sonuç ile gerçek sonuç aynı çıkmadı. Yani test başarısız.

**Blocked – Skipped (Engellendi - Atlanıldı):** Test case çalıştırılamadı. Örneğin uygulamamızda kayıt kısmı çalışmıyor ise kayıt kısmı başarısızdır ama giriş yapma kısmı Blocked – Skipped durumuna girer. Çünkü giriş yapma kısmı kaydolma kısmına bağlıdır ve şu anda giriş yapma kısmını test etmemiz imkansızdır çünkü kayıt kısmı çalışmıyor.

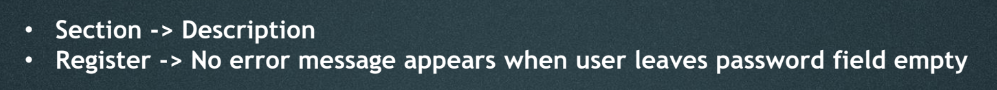
**Bug (Defect) Report Writing (Hata Raporu Yazma)**

🡺 Bug Report (Hata Raporu) bir hatanın oluşumunun, niteliğinin (nasıl olduğunun) ve durumunun belgelenmesidir.

🡺 Bug (Hata), bir iş ürününde, işin gerekliliklerini veya özelliklerini karşılamayan bir kusur veya eksikliktir. Eş anlamlıları **defect** ve **fault** dur.

🡺 Bug Report (Hata Raporu) yazarken doldurmamız gereken bazı önemli alanlar vardır.

**1) Bug Report Title (Hata Raporu Başlığı):** Hata raporu başlığı yazmak için birçok format vardır, ancak hatanın başlığını yazmanın en iyi yolu önce hatanın bulunduğu bölümü yazmak ardından bir ok koyup ardından açıklamayı yazmaktır.



**2) Steps To Reproduce (Yeniden Oluşturma Adımları):** Yeniden oluşturulacak adımlar çok spesifik (belirli) olmalıdır. Hatayı yeniden üretmek için attığımız her adımı söylemelisiniz, ayrıca kullandığınız verileri de girmelisiniz. Örneğin sepet için bir e-posta, şifre, veri, sayı veya değer kullandıysanız, bunu belirtmelisiniz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**3) Expected Result (Beklenen Sonuç):** Test case de olduğu gibi beklenen sonuçtur.

**4) Actual Result (Gerçek Sonuç):** Adımlar yürütüldüğünde oluşan sonuçtur. Test case yazarken bu alanı boş bırakıyorduk. Bunun sebebi test case i yürüttüğümüzde oluşacak sonucu bilmiyorduk fakat Bug Report yazarken bu alan doldurulur. Çünkü adımlar yürütülünce oluşacak sonucu biliyoruz.

**5) Test Environment (Test Ortamı):** Hatanın meydana geldiği ortamı(ortamları) belirtiriz.

**6) Screenshot or Video (Ekran Görüntüsü veya Video):** Ekran görüntüsü tüm ekranı göstermelidir. Web sitesinin URL’sini göstermelidir. Kusurlu alanın etrafında kırmızı bir dikdörtgen olmalıdır. Video, klavyedeki tıklamaları göstermelidir. Bir hatanın statik olması, hatanın gerçekleştiğini kanıtlamak için herhangi bir animasyonun gerçekleşmediği anlamına gelir. Örneğin sayfayı 10 saniye gibi çok fazla zamanda yükleme. Bu hatayı raporlarken ekran görüntüsü eklersek, geliştirici geçen süreyi anlamayacaktır fakat video eklersek geçen zamanı görebilecektir.

**7) Bug Priority (Hata Önceliği):** Hatanın önceliğini belirtiriz. 4 seviye vardır.

**Critical (Kritik):** Örneğin, Giriş çalışmıyor – Ana sayfada uygulama kilitleniyor – Sepette yanlış değer gözüküyor.

**High (Yüksek):** Örneğin, Oturum açma sayfası yavaş yanıt veriyor – Kullanıcı profil resmi ekleyemiyor.

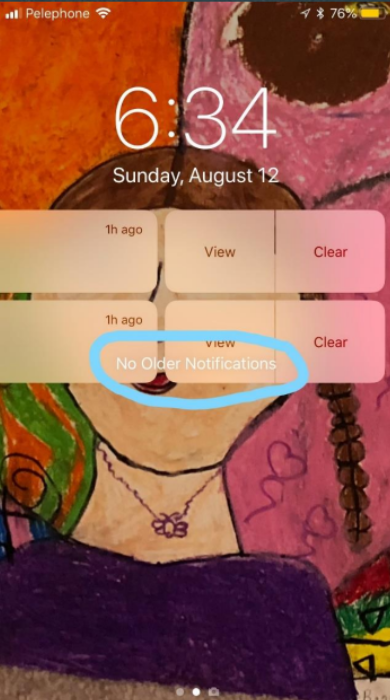
**Medium (Orta):** Örneğin, Bazı sayfaların performansı düşük – Portre modu düzgün çalışmıyor.

**Low (Düşük):** Örneğin, Yazım hataları – Görüntülerin yanlış hizalanması.

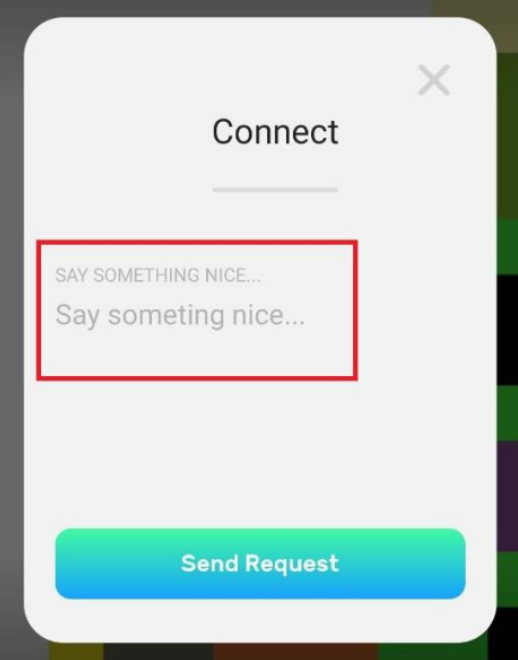
**Defects’ Types (Hata Türleri)**

**1) Functional (İşlevsel):** İşlevsel kusurlarda, uygulamada çalışmayan bir işlev olduğunu kastediyoruz. Örneğin parolamı unuttum işlevi çalışmıyor.

**2) Visual - UI (Görsel - Kullanıcı Arayüzü):** Bu kusurlar, uygulamanın kullanıcı arayüzündeki kusurlardır. Örneğin aşağıdaki resimde bildirimlerimiz var ve üzerlerinde “No Older Notifications” yani “Eski Bildirimler Yok” yazıyor. Bu yazı üzerlerinde değil altlarında olmalıydı.



**3) Content (İçerik):** İçerikte olmaması gereken şeylerin olmasıdır. Örneğin bir uygulamanın Arapça versiyonu içindeki bir kelimenin Arapçaya çevrilmemesi veya aşağıdaki örnekte olduğun gibi bir şeyin 2 kez tekrarlanması (gereksiz şeylerin olması).



**4) Performance (Performans):** Uygulamanın performansında oluşan hatalardır. Örneğin bir video uygulamasında videoların oynatılmasının çok fazla zaman alması veya bir sayfanın çok geç yüklenmesi. Bazı durumlarda, örneğin bir video donduğunda ekranda noktalar dönmeye başlar bu duruma “Lazy Loading” yani “Tembel Yükleme” denir. Eğer tembel yükleme çok uzun sürüyorsa bu da performans hatasına girer.

**5) Suggestion (Öneri):** Öneri, uygulamadaki bir hata değildir, ancak bir test uzmanı olarak sizin bakış açınızdan farklı bir şekilde yapılmasının daha iyi olacağını düşündüğünüz bir şeydir. Örneğin, bir “Place Holder” yani “Yer Tutucu” nun yazı tipinin daha büyük olması gerektiğini söylüyorsunuz.

**Örnek bir hata raporu,**

